1. Qt多线程
2. Qt文本流数据存储

QTextStream文本流类的功能界面和标准的C++的iostream类非常相似。

  iostream和QTextStream的不同点是我们的流操作在一个很容易被继承的QIODevice上，而iostream只能操作一个不能被继承的FILE \*指针。

  我常常实用QTextStream按照自定义格式保存一些数据到文件中，然后根据需要将保存在的数据读取，处理。使用起来比较方便。

1. QFile file("G:/info.txt");
2. **if**(!file.open(QIODevice::WriteOnly | QIODevice::Truncate))
3. {
4. qDebug()<<"文件打开失败："<<file.errorString();
5. **return** -1;
6. }
8. //获取一些信息将并将它保存到文件info.txt中：以获取指定文件下文件信息为例
9. QTextStream cin(&file);
11. //调用文件写入函数，遍历目录将文件信息写入文件
12. writeInfo("F:\\Warcraft III",cin);
13. file.close();
14. Qt曲线图绘制

QCustomPlot是QT下一个方便易用的绘图工具，只有两个文件qcustomplot.h和qcustomplot.cpp组成，将其提升为widget可直接使用。

1. Qt UDP数据传输

例子采用的是单播方式，其他组播和广播也都是一样的代码，稍微修改一下。

首先要写一个发送端。发送端比较简单，只需要几行代码：

当然哦，因为要用到网络，所以这里你需要在pro文件中加入QT+=network，别忘了，执行一下qmake撒，不然是不行的。

main.cpp的代码

**[cpp]** [view plain](http://blog.csdn.net/xuancailinggan/article/details/50519045) [copy](http://blog.csdn.net/xuancailinggan/article/details/50519045)

1. #include <QApplication>
2. #include <QApplication>
3. #include <QUdpSocket>
4. #include <QHostAddress>
6. **int** main(**int** argc, **char** \*argv[])
7. {
8. QApplication a(argc, argv);
9. QUdpSocket \*MySocket=**new** QUdpSocket;//定义一个UDP
11. **for**(**int** i=0;i<100;i++)
12. {
13. QString MyData="hello udpsocket";//要发送的额数据
14. MySocket->writeDatagram(MyData.toUtf8(),QHostAddress::LocalHost,10000);//把数据发送出去,这里的端口你可以随便写，但是要注意它没被占用
15. qDebug()<<MyData;
17. }
19. **return** a.exec();
20. }

如图一个发送端就写好了。

下面是一个接收端，一样的，新建一个项目以后，要添加QT+=network

接收端头文件的代码：

**[cpp]** [view plain](http://blog.csdn.net/xuancailinggan/article/details/50519045) [copy](http://blog.csdn.net/xuancailinggan/article/details/50519045)

1. #ifndef WIDGET\_H
2. #define WIDGET\_H
4. #include <QWidget>
5. #include <QUdpSocket>
6. #include <QHostAddress>
7. #include <QDebug>
9. **class** Widget : **public** QWidget
10. {
11. Q\_OBJECT
12. **private**:
13. QUdpSocket \*read;//因为要在槽函数使用，因此这里在头文件进行定义
15. **public**:
16. Widget(QWidget \*parent = 0);
17. ~Widget();
18. **public** slots:
19. **void** readData();//槽函数用于对接受的数据进行处理
20. };
22. #endif // WIDGET\_H

cpp文件的代码：

**[cpp]** [view plain](http://blog.csdn.net/xuancailinggan/article/details/50519045) [copy](http://blog.csdn.net/xuancailinggan/article/details/50519045)

1. #include "widget.h"
3. Widget::Widget(QWidget \*parent)
4. : QWidget(parent)
5. {
6. read=**new** QUdpSocket;
7. read->bind(10000,QAbstractSocket::DontShareAddress);//绑定端口
8. connect(read,SIGNAL(readyRead()),**this**,SLOT(readData()));//这里要注意，在接收到数据以后，readready()函数会被自动触发，你可以理解成按钮的click事件
9. }
10. **void** Widget::readData()
11. {
12. **while** (read->hasPendingDatagrams())
13. {
14. QByteArray MyDate;//因为传送来的数据类型是未知，所以用bytearray
15. MyDate.resize(read->pendingDatagramSize());//mydata的数据大小取决于接收到的数据
16. read->readDatagram(MyDate.data(),MyDate.size());//读取数据
18. qDebug()<<QString::fromUtf8(MyDate);
19. }
20. }
22. Widget::~Widget()
23. {
25. }

程序运行时候，要先运行接收端，再运行发送端。

**QT中的组播**

因为之前一直听说组播，也没有实际用过，组播的原理大致就是服务器往某一组播地址和端口发数据，之后客户端从指定的组播地址和端口去取数据，好处就是减轻了服务器发送的压力，弊就是只要是加入到组播组的用户都可以收到改数据，如果往同一组播地址和端口发送数据的话，可能会引起风暴。   
具体代码如下：

QUdpSocket \*revWBudp;

revWBudp = new QUdpSocket;

//绑定本地端口

revWBudp->bind(QHostAddress::AnyIPv4, multicastPort, QUdpSocket::ShareAddress);

//加入组播组

bool result = revWBudp->joinMulticastGroup(QHostAddress(multicastIP));

if(result)

{

//设置缓冲区

revWBudp->setSocketOption(QAbstractSocket::ReceiveBufferSizeSocketOption,1024\*1024\*8);

//连接接收信号槽

connect(revWBudp,SIGNAL(readyRead()),this,SLOT(dataReceived()));

｝

其余操作跟普通udp连接类似。

1. 数据转换计算
2. python 数据库
3. python 界面设计
4. python 开关门逻辑设计